Simplicity in IT: The Power of *Less*

Tiziana Margaria, *Universität Potsdam, Germany*
Mike Hinchey, *Lero—the Irish Software Engineering Research Centre*

Simplicity is a mindset, a way of looking at solutions, an extremely wide-ranging philosophical stance on the world, and thus a deeply rooted cultural paradigm. The culture of "less" can be profoundly disruptive, cutting out existing "standard" elements from products and business models, thereby revolutionizing entire markets.

Although a software system is immaterial and not an object per se, many of its aspects embody beauty as simplicity: the design or concept's efficiency, the structure's purity and clarity, the code's stringent organization, the running artifact or behavior's reliability, and the system's usability, its versatility, and its longevity as it morphs over time. And yet, the beauty of simplicity is easily spoiled by poor decisions, misguided features, and outdated assumptions about efficiency capstones. Particularly in IT, with its ethereal nature, simplicity is relegated to the inner layers, and too often remains unseen, implicit, and largely underappreciated.

DOING MORE WITH LESS

Modernization and progress often come from eliminating components or elements, defining novelty through "*-less" properties—think horseless carriages, bagless vacuum cleaners, cordless telephones, wireless anything, and more.[1]

When more features means more profits, embracing simplicity and fewer features as a guiding strategy is, at first, risky and unpopular. However, simplification can take several forms. Ease of use by end users is the most obvious as it is at the skin of an IT product; it has been well studied by the human-computer-interface community and we do not address it here.

Simplifying the deeper layers of IT, beneath the pure user interface, has so far been largely neglected, yet this offers an enormous potential for consolidation, unification, and streamlining, as well as for exploitation and use beyond even the long design, production, and maintenance chain. Often, awareness of these "deep IT" contributions to the essence and qualities of a system remains confined within the IT specialists' community, and is therefore unknown to the IT end users. The path to exploitation, once an invention has been seeded, is often itself a matter of highly inventive and innovative streamlining and simplification, where less leads to more.

For a theory or an initial prototype to make a difference in an industry, a market, and ultimately to society is a long path with enormous simplification and innovation potential, but simplification is still undervalued as a specific research goal.

Model-based testing (MBT) is one recent stunning success story in which the simplification and streamlining connected with uptake play a pivotal role.[2] MBT initially suffered from requiring á priori test models, rarely available in practice because they are expensive to construct and maintain throughout a system's life cycle. However, research revealed that adequate test models could be constructed fully automatically on the fly via automata learning techniques.[3] This essentially turns MBT into a test-based modeling discipline. This step, which has the potential to drastically impact today's testing technology, is by no means obvious and, a decade later, still requires significant engineering effort to prove its promises. For example, the underlying automata learning machinery needs further enhancement to be sufficiently scalable.[4] Thus the simplification potential impacting the entire software design life cycle is realized here by adopting an *upside-down perspective* between models and testing, in combination with algorithmic research and solid engineering to fix the loose ends. By also turning theory-intensive and exotic automata learning into a practical discipline,[5] this endeavor can help make MBT applicable as mainstream technology.

Even if the quest for simplicity starts incrementally with those changes that bring the most direct benefit, it will eventually impact larger aspects as well.

The central questions in dealing with simplicity in deep IT are what is essential for the purpose, and what, in the realization of this essential functionality, can be achieved with less infrastructure, with less indirection, with lower costs, and, ultimately, with less risk?

The move to *less* presupposes the ability to smoothly adapt and change. In this regard, architectural design is still a bottleneck; changing a system's architecture is painful, as is modifying an information system schema. Even if the quest for simplicity starts incrementally with those changes that bring the most direct benefit, it will eventually impact larger aspects as well, leading to a reprioritization of decisions and values that might also affect domains, such as the system architecture, which at first seem untouchable.

IN THIS ISSUE

The first two articles in this special issue concern the big picture in describing and adopting a simplicity framework in IT.

In "ITSy—Simplicity Research in Information and Communication Technology," Barry Floyd and Steve Bosselmann summarize the state of the art and the research issues in the area of simplicity and IT, spanning literature review, cultural principles, and a number of recommendations for a wider spread of the simplicity culture in IT that is the focus of the ITSy European Support Action project (www.cs.uni-potsdam.de/sse/ITSy).

In "Achieving Simplicity with the Three-Layer Product Model," Jan Bosch addresses the delegation issue in IT product design and maintenance. From experience with large companies, several principles and guidelines emerge on how to determine the in-house focus of competence for products, and how to deal with the resulting evolution cycle for products that include both in-house and external parts.

The next four articles explore simplicity's potential for success throughout the software design life cycle, tackling different aspects of heterogeneity and variability.

In "Component Models for Reasoning," Cristina Seceleanu and Ivica Crnkovic review an abstract characterization of different widespread component models with respect to their native adequacy for formal reasoning, from the perspective of composition—the fundamental benefit associated with component reuse. Early validation, proof of compatibility and correctness, and property-driven characterization, such as with contracts, are essential to simplify the burden of validation and move it as far to the front of design and development as possible.

In "Variability Management beyond Feature Models," Anna-Lena Lamprecht, Stefan Naujokat, and Ina Schaefer present a behavioral approach to simplify the management and realization of variants that arise in connection with product and process diversification and evolution. This fully automatic generation of process variants promises a knowledge-driven simplification of variant definition from which direct synthesis of the corresponding executable processes becomes feasible.

In "Plug-and-Play Higher-Order Process Integration," Johannes Neubauer and Bernhard Steffen propose simplicity through higher-order process integration as a way to elegantly overcome unexpected hurdles for the integration, variability, and interoperability of business processes. This new way of involving the application expert in the software development process reduces "communication accidents" from requirements engineering to time to market. The authors present a powerful plug-and-play discipline of higher-order design, wherein processes and services can be moved around just like data.

In "Taking Control of Your Engineering Tools," Chandra Prasad and Wolfram Schulte describe the transition to runtime of large, distributed, and heterogeneous software projects. While traditional software development tools do not support migration to a cloud-based engineering system, the build system created this past year at Microsoft does. Based on their experience with creating this system, the authors outline six principles on how current engineering tools need to evolve to enable simpler and controllable cloud-based development.

Two short contributions round out the picture by providing insights on the state of the art in two distinct application domains.

In "A Tool Integration Framework for Sustainable Embedded Systems Development," Tiberiu Seceleanu and Gaetana Sapienza address the state of the art of tool integration in the embedded system industry, adopting established research principles into practice.

In "Applying KISS to Healthcare Information Technology," Regina Herzlinger, Margo Seltzer, and Mark Gaynor summarize the current state of healthcare IT, sketching issues and challenges due largely to ingrained heterogeneity and poor alignment in departmental information systems. They also propose a selection of simplifying principles that, although they seem obvious to software engineers, are often undermined in real life.

A more democratic, participative, inclusive, and global-IT-supported society is the future vision embraced at the European Community level. Disintermediation—the elimination of specialized intermediaries to the creation of IT artifacts—and programming-less software creation are the new frontiers in which simplicity can truly empower every individual, and potentially drive the cultural change that our society today expects. C

References

1. T. Margaria and B. Steffen, "Simplicity as a Driver for Agile Innovation," *Computer*, vol. 43, no. 6, 2010, pp. 90-92.
2. J. Tretmans, "Model-Based Testing and Some Steps towards Test-Based Modelling," *Formal Methods for Eternal Networked Software Systems*, LNCS, vol. 6659, 2011, pp. 297-326.
3. A. Hagerer, et al., "Model Generation by Moderated Regular Extrapolation," *Fundamental Approaches to Software Eng.*, LNCS, vol. 2306, 2002, pp. 80-95.
4. H. Raffelt, et al., "Dynamic Testing via Automata Learning," *Int'l J. Software Tools for Technology Transfer*, vol. 11, no. 4, 2009, pp. 307-324.
5. M. Isberner, F. Howar, and B. Steffen, "Learning Register Automata: From Languages to Program Structures," *Machine Learning*; forthcoming; doi:10.1007/s10994-013-5419-7.

Tiziana Margaria *is chair of service and software engineering at the Institute of Informatics, Universität Potsdam, Germany. Her research interests include model-based system and service engineering as well as process science for healthcare and life sciences. Margaria received a PhD in computer and systems engineering from the Politecnico di Torino, Italy. She is a member of the board of the European Forum for Information and Computation Science and Technology, a Fellow of the Society for Design and Process Science, and vice president of the European Association of Software Science and Technology. Contact her at margaria@cs.uni-potsdam.de.*

Mike Hinchey *is the director of Lero—the Irish Software Engineering Research Centre and a professor of software engineering at the University of Limerick, Ireland. His research interests include various aspects of software engineering, particularly autonomous systems and formal methods for software-intensive systems. Hinchey received a PhD in computer science from the University of Cambridge, UK. He is a vice president of the International Federation for Information Processing, a senior member of IEEE, and a Fellow of the British Computer Society and Irish Computer Society. Contact him at mike.hinchey@lero.ie.*